The same rules apply to these conversion operators as to placeholder return types of ordinary member functions. The last conversion operator, for example, combines **auto** with **const** and the pointer operator. Note~~, however,~~ that ~~because~~ these operators ~~do not have unique names,~~ ~~either their implementation must be inline within the class (as above) or they must be distinguishable in some other way, e.g., by cv-qualification:~~

```
struct R
{
    operator auto();        // OK, deduced type not known
    operator auto() const;  // OK, const-qualified, deduced type not known
};

R::operator auto() { return "hello"; }  // OK, deduce type const char*.
R::operator auto() const { return 4; }  // OK, deduce type int.

void f2()
{
    R r;

    const char* s = r;  // OK, choose nonconst conversion to const char*.
    int         i = r;  // OK, choose const conversion to int.
}
```

In **struct** R, the two conversion operators can coexist even before their return types are deduced because one is **const** and the other is not. The deduced types must be known before the conversion operators are invoked, as usual.

Use Cases

Complicated return types

In their book *Scientific and Engineering C++*,[4] authors Barton and Nackman pioneered template techniques that are now widely used. They described a system for implementing SI units in which the individual unit exponents were held as template value parameters; e.g., a distance exponent of 3 would denote cubic meters. The type system was used to constrain unit arithmetic so that only correct combinations would compile. Addition and subtraction require units of the same dimensionality (e.g., square meters), whereas multiplication and division allow for mixed dimensions (e.g., dividing distance by time to get speed in meters per second).

We give a simplified version here, supporting three base unit types for distance in meters, mass in kilograms, and time in seconds:

```
// unit type holding a dimensional value in the MKS system
template <int DistanceExp, int MassExp, int TimeExp>
class Unit
```

[4] **barton94**